

Program Reasoning

11. Search Space Pruning

Kihong Heo

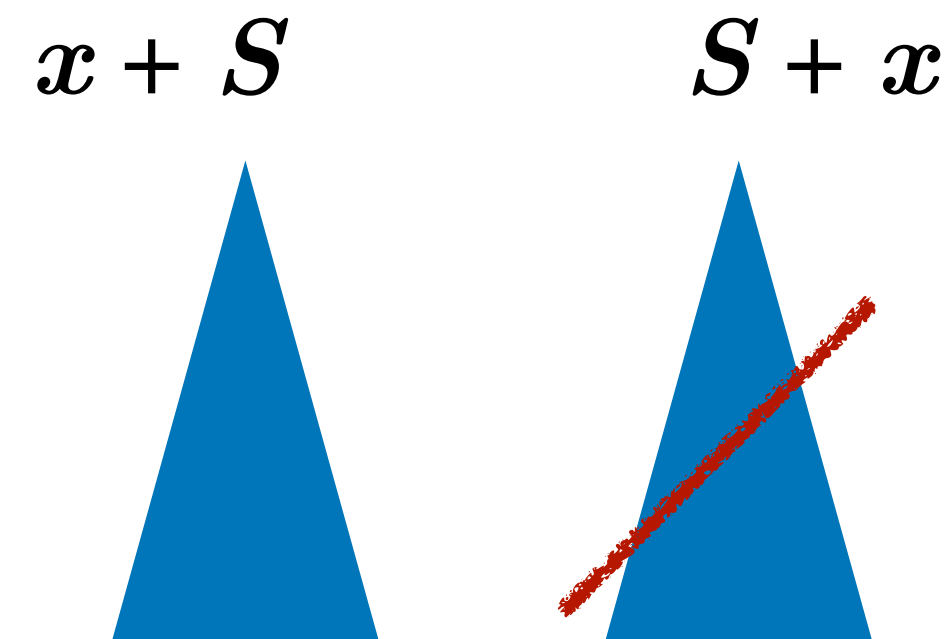


Search Space Pruning

- Problem: too huge search space
- How to prune the search space for program synthesis?
- Common strategies:
 - Equivalence reduction = discard **redundant** candidates
 - Top-down propagation = discard **unpromising** candidates

Equivalence Reduction

- Discard **redundant** subprograms as early as possible
- How to implement this idea for top-down and bottom-up searches?



Eq. Reduction for Top-down

- If two non-ground programs will be expand the same set of programs, discard one of them
- What candidates can be discarded by the following programs in the queue?

`if (true) y S`

`1 * (S + S)`

`sort(S)`

Top-down + Eq. Reduction

```
top-down( $G = \langle \Sigma, N, R, S \rangle, \phi$ ):  
   $Q := \{S\}$   
  while  $Q \neq \{\}$ :  
     $p := \text{dequeue}(Q)$   
    if  $\text{ground}(p) \wedge \phi(p)$ : return  $p$   
     $P' := \text{unroll}(G, p)$   
    forall  $p' \in P'$ :  
      + if not  $\text{equiv}(p, p')$ :  
         $\text{enqueue}(Q, p')$ 
```

```
unroll( $G = \langle \Sigma, N, R, S \rangle, p$ ):  
   $Q' := \{\}$   
   $A := \text{left-most non-terminal in } p$   
  forall  $(A \rightarrow B)$  in  $R$ :  
     $p' := p[B/A]$   
     $Q' := Q' \cup \{p'\}$   
  return  $Q'$ 
```

Eq. Reduction for Bottom-up

- How to apply equivalence reduction for bottom-up search?
- Problem: candidates are all ground but might not be whole
- Solution for PBE: **observation equivalence**
 - We only care of equivalence on the given inputs
- E.g., the programs below are observationally equivalent modulo the inputs (1,0) and (2,1)

x $\text{if } (x \leq y) \ y \ x$

Example

Specification

Find a function $f(x)$ where $f(1) = 3$

Grammar

$$S \rightarrow x \mid 1 \mid -S \mid S + S \mid S \times S$$

Enumeration

iter 1	x	1		
iter 2	$-x$	1	$1 + x$	$1 + 1$
	$x + x$	$x + 1$	$x \times x$...
iter 3	$x + x + x$			

Bottom-up + Eq. Reduction

```
bottom-up( $G = \langle \Sigma, N, R, S \rangle, \phi$ ):  
   $Q :=$  set of all terminals in  $G$   
  while true:  
    forall  $p$  in  $Q$ :  
      if  $\phi(p)$ : return  $p$   
     $Q +=$  grow( $G, Q$ )
```

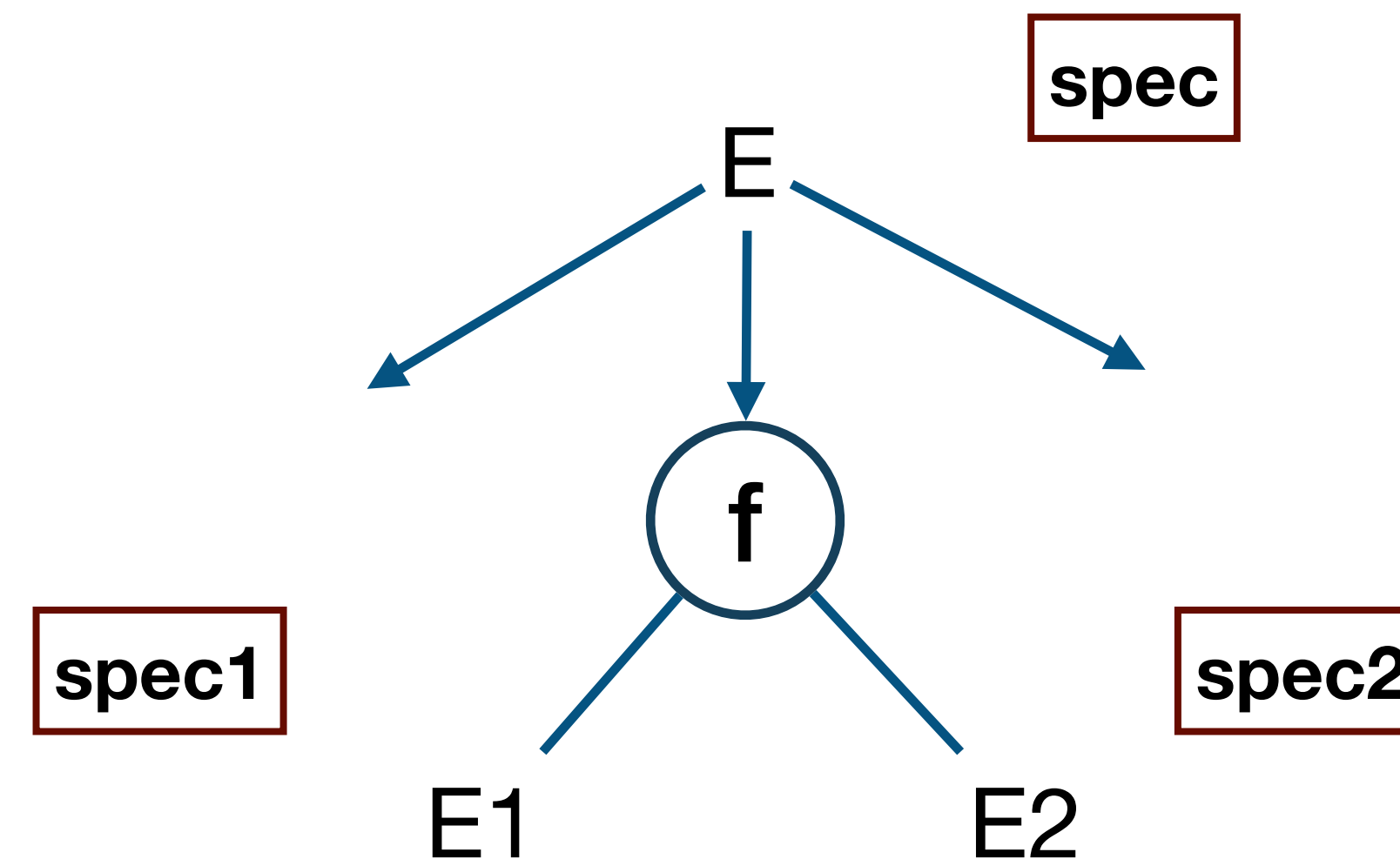
```
grow( $G, Q$ ):  
   $Q' := \{\}$   
  forall ( $A \rightarrow B$ ) in  $G$ :  
     $Q' += \{ B[C \rightarrow p] \mid p \in Q, C \rightarrow^* p \}$   
  +  $Q' := \{ p' \in Q' \mid \text{forall } p \in Q. \neg \text{equiv}(p, p') \}$   
  return  $Q'$ 
```


Search Space Pruning

- Problem: too huge search space
- How to prune the search space for program synthesis?
- Common strategies:
 - Equivalence reduction
 - Top-down propagation

Top-down Propagation

- Discard **unpromising** subprograms as early as possible
- Given a spec and a production, infer specs for subprograms (divide-and-conquer)
 - When $f\langle E_1, E_2, \dots, E_n \rangle (In) = Out$ where E_i is a subprogram
 - What is the spec for each E_i ?
 - If any E_i is undesirable, discard f



Example: String Manipulation

Specification

Find a function $f(x)$ where $f(\text{"SA"}) = \text{"USA"} \wedge f(\text{"AE"}) = \text{"UAE"}$

Grammar

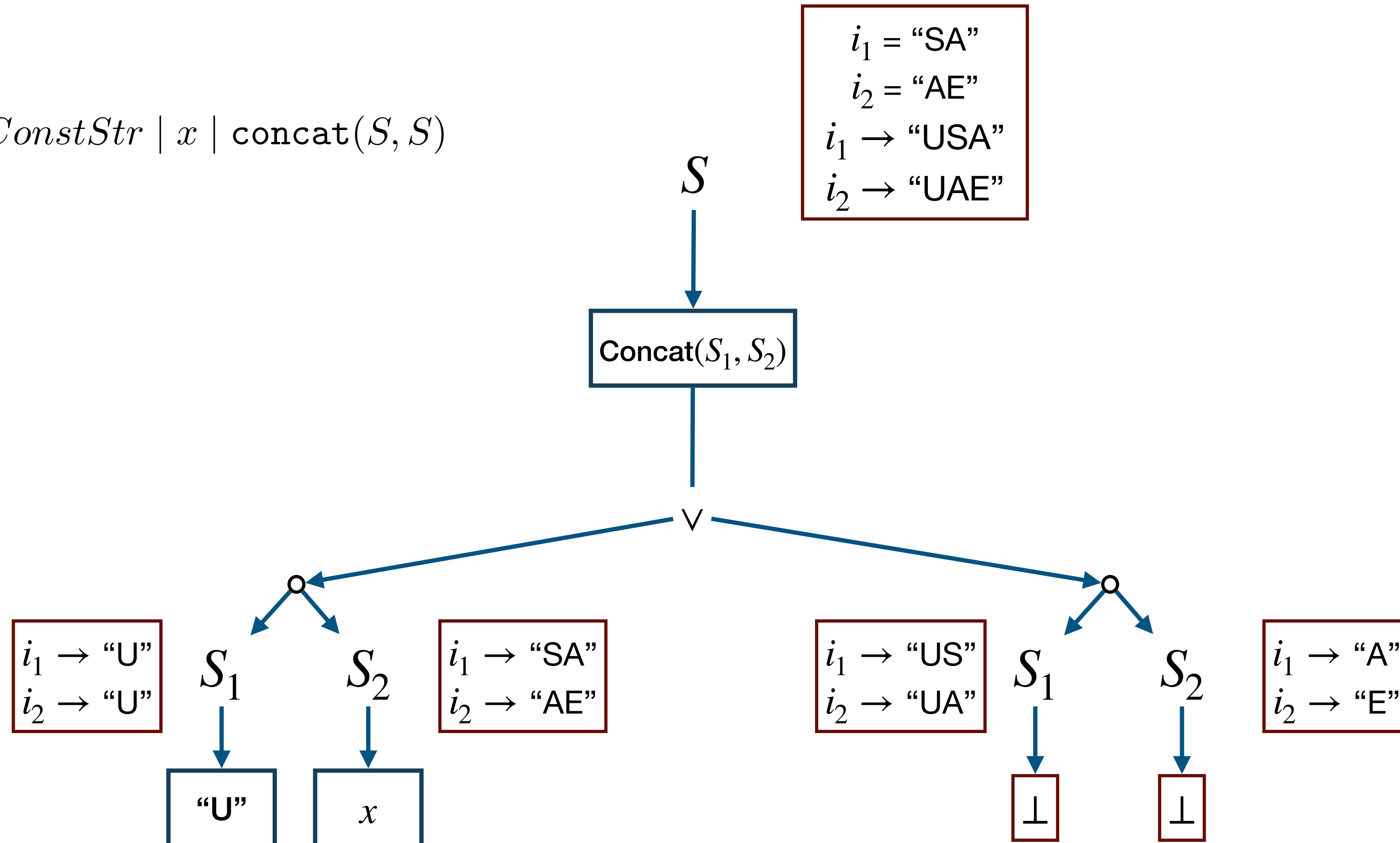
$$S \rightarrow \textit{ConstStr} \mid x \mid \text{concat}(S, S)$$

Examples

$$\text{concat}(\text{"U"}, \text{"SA"}) = \text{"USA"}$$

TDP for String Manipulation

$S \rightarrow ConstStr \mid x \mid \text{concat}(S, S)$



Example: List Manipulation

Specification

Find a function $f(x)$ where $f([1; -3; 1; 7]) = [2; -2; 2; 8]$

Grammar

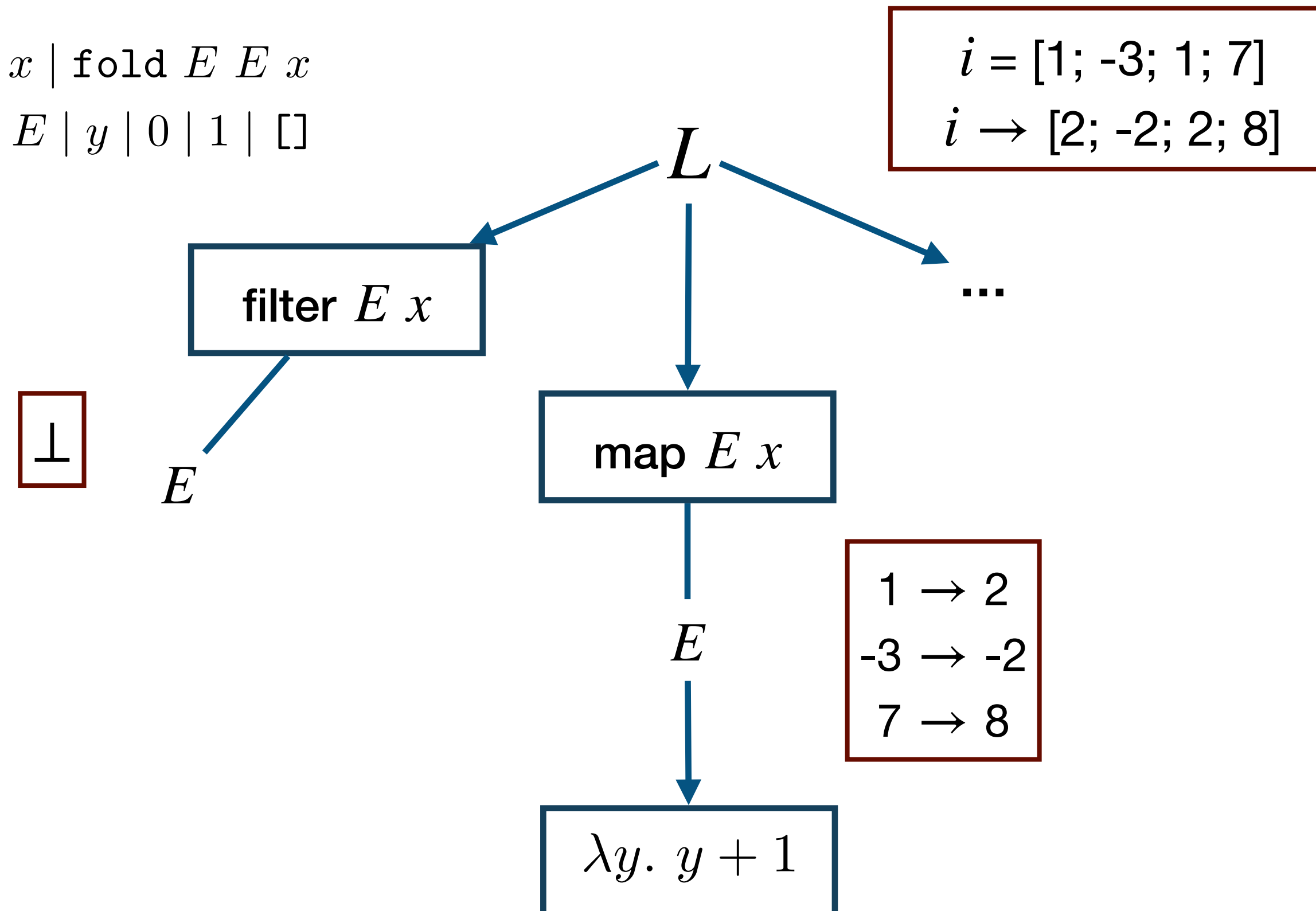
$$L \rightarrow \text{filter } E \ x \mid \text{map } E \ x \mid \text{fold } E \ E \ x$$
$$E \rightarrow \lambda y. E \mid E + E \mid E \leq E \mid y \mid 0 \mid 1 \mid []$$

Examples

$$\text{filter } (\lambda y. y \leq 0) \ [1; -3; 1; 7] = [-3]$$
$$\text{map } (\lambda y. y + 1) \ [1; -3; 1; 7] = [2; -2; 2; 8]$$
$$\text{fold } (\lambda y. \lambda z. y + z) \ 0 \ [1; -3; 1; 7] = 6$$

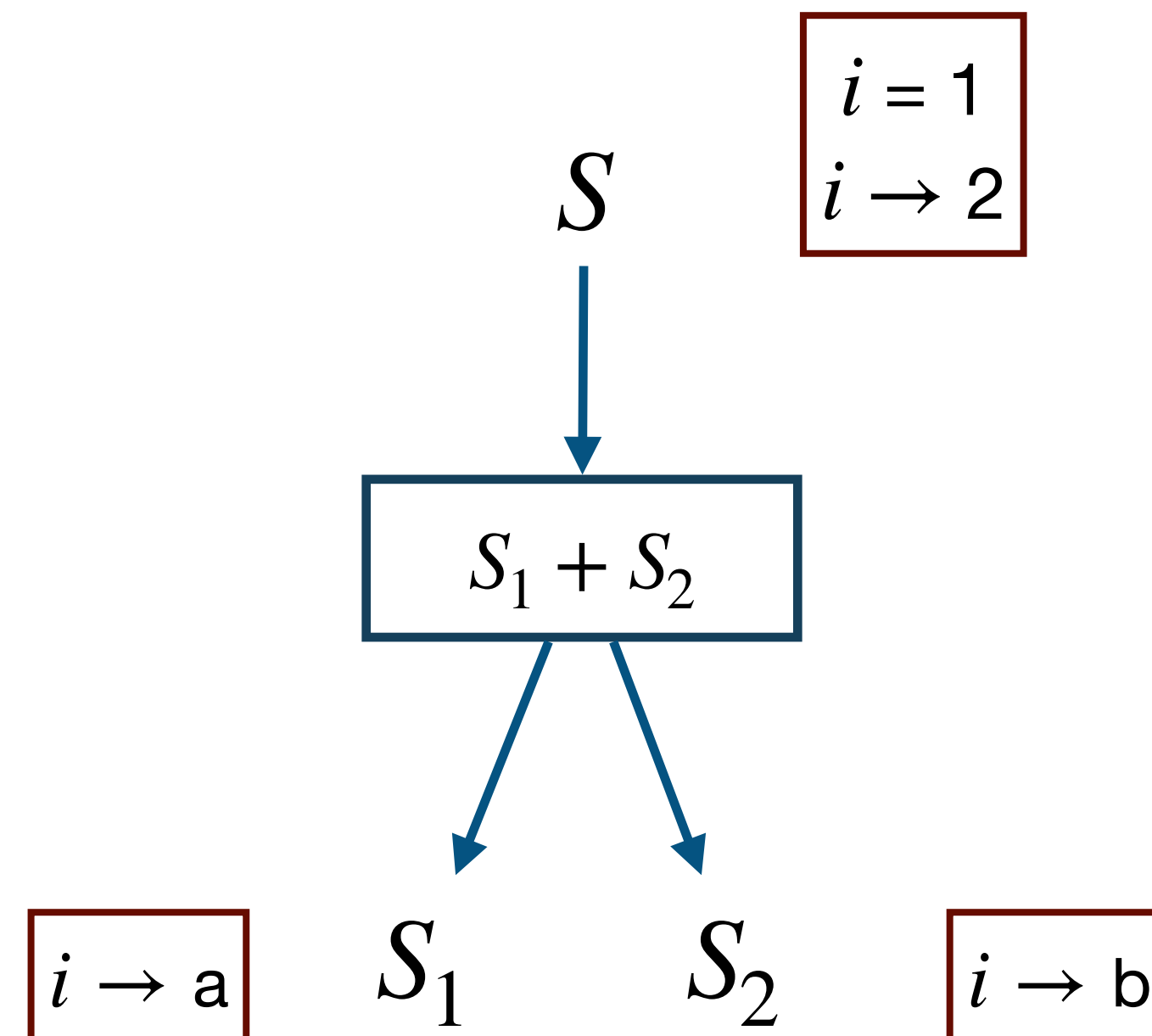
TDP for List Manipulation

$L \rightarrow \text{filter } E \ x \mid \text{map } E \ x \mid \text{fold } E \ E \ x$
 $E \rightarrow \lambda y. E \mid E + E \mid E \leq E \mid y \mid 0 \mid 1 \mid []$



Limitation of TDP

- Applicable only when the function is injective so that the inverse exists
- For example, $(+) : \text{int} \rightarrow \text{int} \rightarrow \text{int}$



What are possible values of a and b?

Summary

- Challenge: huge search space
- Equivalence reduction = discard **redundant** candidates
 - Applicable to top-down and bottom-up searches
- Top-down propagation = discard **unpromising** candidates
 - Applicable to top-down search when the inverse functions are computable